Amendments to the Specification:

Please replace the paragraph beginning at **page 2, line 15**, with the following amended paragraph:

Another type of technique for handling delayed read commands in the queue of the FIFO buffer is known as head-of-list alternation ~~alteration~~. Head-of-list alternation ~~alteration~~ involves an added capability to alternate or swap another read command within the FIFO buffer in place of the delayed read command at the head of the list in the queue. Thus, upon encountering a first delayed read command, and if the next command in the FIFO buffer is also a read command, the relative position of the first delayed command and the next read command is alternated, so that an attempt is made to complete the next read command while the first read command is delayed. After the swap or alternation, completion of the second command is attempted. If the second command is successfully completed, it is unloaded from the queue and completion of the first delayed read command is again attempted. If the first read command is again delayed, the head-of-list alternation will again seek to substitute another read command following the first delayed read command, if another such read command is available in the queue. However, if the next command in the FIFO buffer is not a read command, the first delayed read command is again retried until it is completed. This head-of-list alternation therefore works only if two read commands are available in sequential positions in the queue of the FIFO buffer. If a command other than a read command follows a delayed read command, head-of-list alternation is not possible.

Please replace the paragraph beginning at **page 8, line 17**, with the following amended paragraph:

The command queue 14 establishes a circular queue command buffer 30 of the form shown in Figs. 2A to 2R. Conventional I/O commands 32 are contained within the queue command buffer 30, and the commands 32 are completed in a circular manner in the sense that the commands 32 contained within the buffer 30 are logically completed in a loop so that a first command 32a is typically completed immediately after the last command 32n. Moreover, as a general consequence of the circular and logical loop, the commands are also typically completed in a first-in first-out (FIFO) sequence. However, as will be seen from the following description of the present invention, delayed read commands in a loop may not strictly be completed in a FIFO sequence, although their completion will occur in a more efficient manner than if a strict FIFO sequence was ~~oberved~~observed.

Please replace the paragraph beginning at **page 10, line 21**, with the following amended paragraph:

The commands 32 are placed in the buffer 30 by the conventional enqueuing logic 18, shown in Fig. 1. The enqueuing logic 18 includes a put pointer (not shown) which points to a location in the buffer 30 of an invalid command. An invalid command is one that has been previously completed. An invalid command, having been completed, can be replaced by a new, valid command for which subsequent completion is desired by the I/O device 20. The put pointer of the enqueuing logic 18 operates independently of the get queue pointer ~~32~~34, the loop start pointer 36 and loop end pointer 38 (Figs. 2A to 2R). The put pointer is generally in advance of the location of the get queue pointer ~~32~~34, to enable the buffer 30 to be loaded with commands which are to be completed subsequently. However, when there are no valid commands in the buffer 30, the get queue pointer 34 and the ~~end~~ put pointer~~s~~ will be at the same location, with the put pointer identifying the location where the enqueuing logic 18 will insert the next valid command and with the get queue pointer 34 identifying that same location where the queue processor 10 will start to work on the next valid command entered at that location. The enqueuing logic 18 manages the location of the put pointer in ~~an~~ a conventional manner. Consequently, the activity of the put pointer and the enqueuing logic 18 is not ~~not~~ a part of the present invention, although the present invention does interact with this conventional functionality in the manner described below.

Please replace the paragraph beginning at **page 14, line 19**, with the following amended paragraph:

At step 60, the loop flag is set, which indicates that the present invention is active. Also at step 60, the loop start pointer 36 is set at the position of the first delayed read transaction, which at this stage of the example, is the command 32a at the location of the get queue pointer 34, as shown in Fig. 2C. The loop end pointer 38 is set at the next position in the command buffer 30, which is command 32b in this example. Thus, as a result of executing step 60, the loop start and end pointers 36 and 38 are set on the first delayed and next sequential read commands 32a and 32b, respectively, as shown in Fig. 2C. The process flow then advances the get queue pointer at step 61. At step 61, the get queue pointer 34 is advanced to the next sequential read command ~~32c~~ 32b, which is shown in Fig. 2D as command 32b. The advancement of the get queue pointer 34 at step 61 is accomplished by the normal functionality of the bus interface controller 12 (Fig. 1).

Please replace the paragraph beginning at **page 15, line 1**, with the following amended paragraph:

At this point in this example, some of the process flow 40 repeats with respect to the read command 32b (Fig. 2D). The command 32b is determined to be valid at 44, because the determination at 58 is only affirmative when the next read command is a valid read command. The data mover is started at 50 with respect to the read command 32b (Fig. 2D), and a wait loop is established at 52 until the data mover is done. At 54, the response from the target device is evaluated to determine whether the read command ~~32c~~ 32b has been delayed. For purposes of this example, assume that the read command ~~32c~~ 32b is delayed, causing the process flow 40 to advance to 56. The determination at 56 is affirmative, because the loop flag has previously been set at step 60. Consequently, the process flow advances from 56 to the determination at 62.

Please replace the paragraph beginning at **page 17, line 11**, with the following amended paragraph:

Because the second read command 32b was completed, the read command 32b is marked or set invalid at step 70. The read command ~~32c~~ 32b is shown as invalid in Fig. 2I as a result of executing step 70. A determination is made at step 72 of whether the loop flag is set, which is the case because the loop flag will remain set since the read commands 32a and 32c are still delayed at this point in this example. It is thereafter determined at step 74 whether the get queue pointer 34 is at the location of the loop start pointer 36. As shown in Fig. 2I, this is not the case, so the process flow advances to step 76. At step 76, it is determined whether the loop start pointer 36 is equal to or at the location of the loop end pointer 38. In this example, the determination at 76 is also negative, as shown in Fig. 2I, so the process flow advances to step 62. The get queue pointer 34 is not at the location of the loop end pointer 38, so the determination at 62 is negative. The process flow 40 moves to step 61 where the get queue pointer 34 is advanced to the location of the third delayed read command 32c, as shown in Fig. 2J.

Please replace the paragraph beginning at **page 19, line 24**, with the following amended paragraph:

With the loop flag cleared at step 80, the loop start pointer 36 and the loop end pointer 38 are eliminated, as shown in Fig. 2P, to abolish the loop of delayed read commands. The process flow thereafter moves to step 61 where the get queue pointer 34 is advanced one location to the third delayed read command 34c, as shown in Fig. 2P. The third delayed read command 32c is still valid, as determined at 44, and is attempted to be completed by starting the data mover at 50. After the read command 32c has been completed as determined at 52, it is determined at 54 that the third read command transaction 32c is completed. The completed third read command 34c is set invalid at 70, as shown in Fig. 2Q. The determination at step 72 indicates that the loop flag is no longer set, causing the process flow to move to step

61 where the get queue pointer 34 is advanced to the next command ~~34d~~ 32d, which in this example is the non-read command 32d, as shown in Fig. 2R.